Práctica 7:

GRAFOS

OBJETIVO

- Implementación de grafos en C++.
- Implementar y comprender el algoritmo de Dijkstra.

MATERIAL A ENTREGAR

- *.h, *.cpp
- makefile
- costes.png
- Ejercicios apartado 1. **(previo)**

DESARROLLO

La Generalitat Valenciana está intentando mejorar la eficiencia en la extinción de incendios forestales de cara al próximo verano. Para ello nos ha encargado realizar el estudio para la sierra de Espadán. En dicha sierra se encuentran 2 retenes de bomberos (identificados como 0 y 1) para los diferentes pueblos de la zona. Nuestra misión es decidir para cada pueblo, qué retén deberá acudir y cuantos kilómetros tendrá que recorrer. Utiliza el algoritmo de **Dijkstra** para resolver este problema.

1. Ejercicios previos

Escribe el algoritmo de **Dijkstra (mejorado)** extraído de las transparencias de clase para un grafo con los nodos identificados de **0** a **n-1**. Deberá estar escrito en C++ o en pseudocódigo cercano a éste, excepto la cabecera, que deberá estar escrita en C++.

Calcula su coste **mejor** y **peor** en **accesos al grafo**.

2. Clase Grafo

Implementar la clase Grafo a partir de las transparencias de clase.

Para representar el grafo de **n** nodos utilizaremos una matriz de adyacencia, es decir, una matriz cuadrada de n x n reales, donde cada posición (i,j) de la matriz representa el peso del arco desde el nodo i hasta el nodo j.

La matriz de adyacencia se creará de forma dinámica mediante un vector de vectores, ambos de la STL. Por tanto, en el constructor de la clase se deberá realizar un *resize* de todos los vectores con el tamaño adecuado.

Para consultar o dar valor a los arcos de un grafo **g**, usaremos el operador paréntesis, g**(i,j)**, donde **i** es el nodo de inicio y **j** el nodo final del arco. Los accesos fuera de rango deberán producir una excepción.

Cada instancia de la clase grafo, guardará en un atributo el **número de accesos** que se realizan sobre ella. Y el método: *unsigned getAccesos()* permitirá acceder a este valor.

3. Creación del Grafo

Crear un método de la clase Grafo que permita crear un grafo no dirigido aleatorio, totalmente conectado (todos los nodos están conectados con todos). El prototipo será el siguiente:

void creaGrafoND(float pmin, float pmax);

donde *pmin* es el peso mínimo que puede tener un arco y *pmax* el peso máximo.

Para el problema que queremos resolver, la distancia mínima entre dos pueblos será de 3 Km y la máxima de 30 Km.

4. Algoritmo de Dijkstra

Implementa el algoritmo de Dijkstra que has diseñado en el apartado 1.

Para comprobar que todo funciona correctamente, se deberá escribir el programa principal *test_dijkstra.cpp*, que deberá realizar lo siguiente:

1. Crear un grafo de 6 nodos
2. Crear un arco dentro del grafo y otro que esté fuera de rango para comprobar si se lanza la excepción. Capturar la excepción para que el programa no pare.
3. Crear un grafo aleatorio con el método *creaGrafoND*. Mostrar el grafo por pantalla.
4. Ejecuta el algoritmo de Dijkstra sobre este grafo.
5. Muestra el resultado obtenido y **comprueba que es correcto**.

5. Solución del problema

Implementa la solución al problema comentado en la introducción para un grafo aleatorio de 10 nodos (*bomberos.cpp*). Recuerda que los nodos 0 y 1 son los retenes de bomberos y el resto los pueblos. Entre los nodos 0 y 1 deberá haber una distancia fija de 50 Km. El resto de distancias entre nodos variará, como se ha comentado, entre 3 Km y 30 Km.

Por pantalla se deberá mostrar primero el grafo aleatorio. A continuación, para cada pueblo, el camino a seguir desde el retén a éste y la distancia total.

6. Cálculo del coste

Escribe el programa *costes.cpp* que analiza el **coste medio** del algoritmo para grafos de talla 5 a 100 en incrementos de 5. Representa el resultado en una gráfica (*costes.png*) **junto con los costes teóricos** obtenidos en el apartado 1.

¿Qué conclusiones sacas de la gráfica obtenida? Escribe tus conclusiones como un comentario del programa.